Theoretical Computer Science 36 (1985) 319-331
North-Holland

EMBEDDINGS OF BINARY TREES IN LINES

Krzysztof DIKS
Instytut Informatyki, Uniwersytet Warszawski, PKiN, 00-901 Warszawa, Poland

Communicated by G. Mirkowska
Received April 1984

Abstract. We show that any n-vertex binary tree can be embedded in a line with dilation-cost $O(n/\log_2 n)$. A provided proof is fully constructive.

1. Introduction

Let $G=(V, E)$, $G'=(V', E')$ be graphs and $|V| \leq |V'|$.[1]
An *embedding* of the graph G in the graph G' is an injection from V to V'.
The *dilation cost* of the embedding F of G in G' is the number

$$\mathrm{DIL}_F(G, G') \stackrel{\mathrm{df}}{=} \max_{(x,y)\in E} d_{G'}(F(x), F(y)),$$

where $d_{G'}(u, w)$ denotes the distance in G' between u and w.

The graph embedding problem is the problem of embedding G in G' in a way that minimizes the dilation cost as much as possible. Graph embeddings appear in various computational problems. (For an overview, see [3].) This paper handles about embeddings between two specific families of graphs occurring in these problems: lines and binary trees.

Lines. An n-vertex line is the graph $L_n=(\{1, 2, \ldots, n\}, \{(i, i+1): i=1, \ldots, n-1\})$.

Binary Trees. A binary tree is a finite set of nodes that either is a single node or consists of a root R and an edge between R and the root of each of two binary trees called the left and right subtree of the root. We shall denote the class of n-vertex binary trees by $\mathcal{T}_n$.

Now we present results of studies of embeddings between lines and binary trees. We have the following from [4].

(1) For each $T \in \mathcal{T}_n$ there is an embedding F of L_n in T with $\mathrm{DIL}_F(L_n, T) \leq 3$.
(2) Let T be a height-h complete binary tree, $n=2^{h+1}-1$.

[1] Terminology is from [1].

Any embedding of T in L_n has dilation cost $\geq (n-1)/(2\lfloor \log_2 n \rfloor)$.

In this paper we complete these results showing that any $T \in \mathcal{T}_n$ can be embedded in L_n with dilation cost $O(n/\log_2 n)$. A provided proof is fully constructive.

2. k-Partitions of graphs

Let $G=(V, E)$ be a graph, $|V|=n$, and let k be an integer, $0<k\leq n$. A *k-partition* of the graph G is a partition of V into l pairwise disjoint and not empty sets V_1, $V_2, \ldots, V_l$ such that $|V_i| \leq k$ for each $i=1, \ldots, l$ and $V_1, V_2, \ldots, V_l$ satisfy the following condition:

(*) $\forall (u, w) \in E$ if $u \in V_i$ then $w \in V_{i-1}$ or $w \in V_i$ or $w \in V_{i+1}$ where $V_0 = V_{l+1} = \emptyset$.

We shall call $V_1, V_2, \ldots, V_l$ the k-partition's sets of G.

Lemma 2.1. *If G has a k-partition, then there is an embedding F of G in L_n with* $\mathrm{DIL}_F(G, L_n) \leq 2k-1$.

Proof. Let $V_1, V_2, \ldots, V_l$ be the k-partition's sets of G. If we define F so that

$$F(V_i) = \{|V_1| + \cdots + |V_{i-1}| + 1, \ldots, |V_1| + \cdots + |V_i|\} \quad \text{for each } i = 1, \ldots, l,$$

then $\mathrm{DIL}_F(G, L_n) \leq 2k-1$. □

3. Embeddings of complete binary trees in lines

We define the natural function $s: \mathbb{N} \to \mathbb{N}$ ($\mathbb{N}$ is the set of natural numbers) as follows:

$$s(n) \stackrel{\mathrm{df}}{=} \min\{2^r : (r+2)2^r \geq n\}.$$

Lemma 3.1. $s(n) = \theta(n/\log_2 n)$.

Proof. In order to prove this lemma it will be sufficient to show that, for each $n > 16$, $n/\log_2 n < s(n) < 4n/\log_2 n$.

The first inequality follows from the fact that

$$(n/\log_2 n)(\log_2(n/\log_2 n)+2) = n(1-(\log_2 \log_2 n - 2)/\log_2 n) < n$$

for each $n > 16$.

To prove the second inequality we notice, that

$$(2n/\log_2 n)(\log_2(2n/\log_2 n)+2) = 2n(1-(\log_2 \log_2 n - 3)/\log_2 n) > n$$

for each $n \geq 2$. And as there is the natural number l such that $2n/\log_2 n \leq 2^l < 4n/\log_2 n$, it then follows from the definition of $s(n)$ that $s(n) \leq 2^l < 4n/\log_2 n$. □

Lemma 3.2. *Let us assume that* $n = 2^{h+1} - 1$, *for some integer* $h \geq 0$. *Then there is an* $s(n)$*-partition of an n-vertex, complete binary tree.*

Proof. W.l.o.g. we may assume that $n > 1$. Let $s(n) = 2^r$ for some $r > 0$. Consider the n-vertex, complete binary tree T as it is shown in Fig. 1.

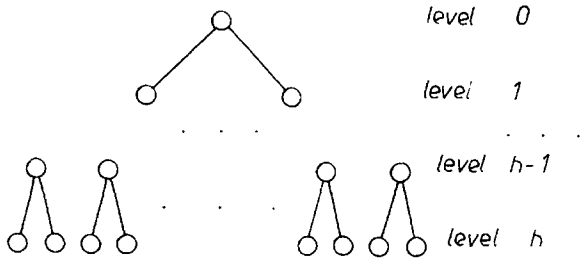


Fig. 1. The tree T.

We construct an $s(n)$-partition of T as follows. Let the first 2^r vertices from the level h, looking from left to right, belong to the set V_1. This is possible because $2^r \leq 2^h$.

Let us assume that we have already formed the sets $V_1, \ldots, V_i$ for some $i < r+1$. All fathers of vertices from the set V_i will belong to V_{i+1}. If we have not already scoooped out all vertices from level h, then the next consecutive 2^{r-1} vertices from this level, looking from left to right, will belong to V_{i+1}. Because 2^{r-1} divides 2^h entirely, the construction of V_{i+1} is correct. If we have formed $V_1, V_2, \ldots, V_{r+1}$ and there are T vertices not belonging to any of these sets, then V_{r+2} will contain these vertices.

Let us note:

(1) if $v \in V_i$ for some i, $1 \leq i \leq r+1$, and
- if v is not the root of T, then the father of v belongs to V_{i+1},
- if v is not a leaf, then sons of v belong to V_{i-1},

(2) if $v \in V_{r+2}$ and
- if v is not the root of T, then the father of v belongs to V_{r+2},
- sons of v belong to $V_{r+2} \cup V_{r+1}$.

We conclude therefore that $V_1, V_2, \ldots, V_k$ ($k = r+1$ or $k = r+2$) satisfy condition (*). In order to prove that it is an $s(n)$-partition of T we should still show that $|V_i| \leq 2^r$ for each $i = 1, \ldots, k$.

Observe that if $|V_i| = l$, then $|V_{i+1}| \leq 2^{r-1} + \frac{1}{2}l$ for each i, $1 \leq i < r+1$. Because $|V_1| = 2^r$, we have $|V_i| \leq 2^r$ for each $i = 1, \ldots, r+1$.

It still remains to be proved that if $k = r+2$, then $|V_{r+2}| \leq 2^r$. To do this, we consider two cases.

Case (a). $|V_{r+1}| = 2^r$. In this case, for each $i = 1, \ldots, r+1$, $|V_i| = 2^r$ and because $(r+2)2^r \geq n$ and $|V_1| + \cdots + |V_{r+1}| + |V_{r+2}| = n$, we have $|V_{r+2}| \leq 2^r$.

Case (b). $|V_{r+1}| < 2^r$. Observe that in this case all vertices from level h belong to $V_1, V_2, \ldots, V_m$, for some $m < r+1$. Consider the subtree of T of vertices from V_{r+2}.

Vertices from V_{r+1} are external ones in this subtree.

Hence, $|V_{r+2}| = |V_{r+1}| - 1 < 2^r$. □

We shall denote the depth of the vertex v in the tree T by $\text{depth}_T(v)$. Assume that $V_1, V_2, \ldots, V_k$ ($k = r+1$ or $k = r+2$) are the $s(n)$-partition's sets of the complete, n-vertex binary tree T obtained by the algorithm from the proof of Lemma 3.2, where $s(n) = 2^r$. Let $w_1^i, w_2^i, \ldots, w_{l_i}^i$ be vertices of V_i, $|V_i| = l_i$, ordered with the preorder. Corollaries 3.3–3.5 immediately follow from the construction of $V_1, V_2, \ldots, V_k$.

Corollary 3.3. *The vertices of each* V_i, $1 \leq i \leq r+1$, *satisfy the condition*

$$\forall 1 \leq j < l_i: 1 \geq \text{depth}_T(w_{j+1}^i) - \text{depth}_T(w_j^i) \geq 0.$$

Corollary 3.4. *If* $k = r+2$, *then vertices from* V_{r+1} *are external vertices for the subtree* T *of vertices from* V_{r+2}.

Corollary 3.5. *If* $v \in V_i$ *for some* i, $1 \leq i \leq r+1$, *and*

- *if* v *is not the root of* T, *then the father of* v *belongs to* V_{i+1},
- *if* v *is not a leaf, then sons of* v *belong to* V_{i-1}.

Let us summarize this paragraph with the following theorem.

Theorem 3.6. *There is an embedding of* n*-vertex, complete binary tree in* L_n *with dilation cost equal to* $O(n/\log_2 n)$.

Proof. The proof can be concluded from Lemmas 2.1, 3.1, and 3.2. □

4. Embeddings of binary trees in lines

Now we shall describe the method of embedding of an arbitrary, n-vertex binary tree in L_n with dilation cost $O(n/\log_2 n)$. Let us assume that $n = 2^{h+1} - 1$ for some $h > 0$ and let T be an n-vertex binary tree. W.l.o.g. we can assume that for each vertex v in T the number of vertices in the right subtree of v is at most equal to the number of vertices in the left subtree of v. We shall transform the tree T to the height-h complete binary tree T' such that an $s(n)$-partition of T' will be simultaneously the $s(n)$-partition of T. Fig. 2 shows the trees T and T' and their corresponding $s(n)$-partitions.

Let R be the root of T. Consider the height-h subtree of T rooted at R. If this subtree is not complete, we add absent vertices to obtain a height-h complete subtree. We shall call these additional vertices virtual. Augmented in this way, tree T will be denoted by GT and its height-h subtree by ST. Fig. 3 shows trees T, GT, and ST.

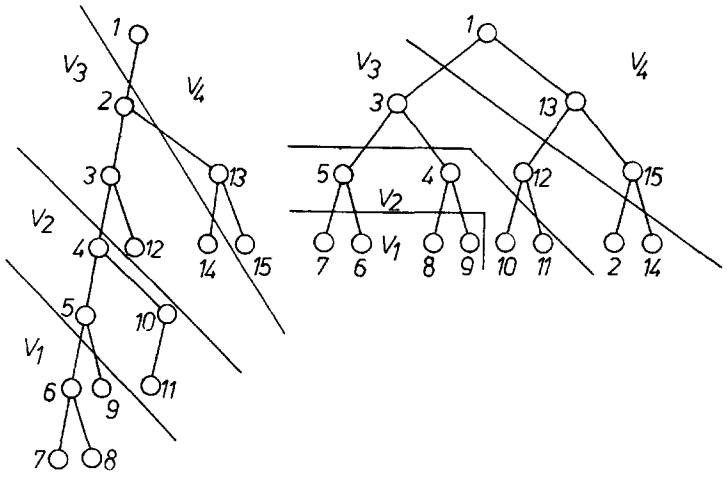


Fig. 2. The trees T and T' and their $s(n)$-partitions.

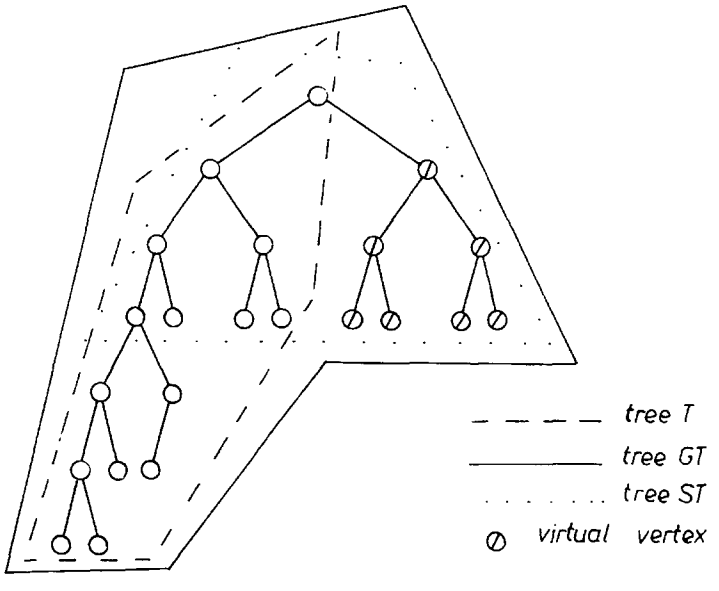


Fig. 3. The trees T, GT, and ST.

Observe that the number of vertices in ST is equal to n. It follows from Lemma 3.2 that there is an $s(n)$-partition of ST. Let $V'_1, V'_2, \ldots, V'_k$ be the $s(n)$-partition's sets of ST constructed by the algorithm from the proof of Lemma 3.2. We shall transform GT (simultaneously T and ST) so as to replace virtual vertices of ST by vertices from T. In the end of this process all ST's vertices will be from T and the sets V'_1, $V'_2, \ldots, V'_k$ of the $s(n)$-partition of ST will be simultaneously the $s(n)$-partition's sets of T. In this process, GT and ST will be changing, but always

- ST will be a height-h subtree of GT rooted at R and augmented to a complete tree by virtual vertices,
- $V'_1, V'_2, \ldots, V'_k$ will be $s(n)$-partition's sets of a current ST.

Now we shall present the algorithm of a construction of sets $V_1, V_2, \ldots, V_k$ of a T's $s(n)$-partition.

begin comment find-$s(n)$-partition algorithm;
$V_k := \{R\}$; **comment** $R \in V'_k$;
$i := k$;
repeat
A { let $w_1, w_2, \ldots, w_l$ be vertices of ST, ordered with the preorder, that belong to V_i and that have sons (in ST) not belonging to V_i;
comment $w_1, w_2, \ldots, w_l$ are vertices from T;
$j := 1$;
B { **while** $j \leqslant l$ **do**
if w_j has a virtual son (in ST) **then**
comment transformation of GT (simultaneously ST and T) such that a virtual son of w_j is replaced by a vertex from T not belonging to $V_k \cup V_{k-1} \cup \cdots \cup V_i$;
GTTRANSFORMATION
else $j := j+1$;
comment all sons of $w_1, w_2, \ldots, w_l$ are from T and do not belong to $V_k \cup V_{k-1} \cup \cdots \cup V_i$;
C { **if** $V_i \neq V'_i$ **then** $V_i := V_i \cup \{w$: w is a son (in ST) of some w_m, $1 \leqslant m \leqslant l$, and $w \notin V_i$ and $w \in V'_i\}$
else begin
$V_{i-1} := \{w$: w is a son (in ST) of some w_m, $1 \leqslant m \leqslant l$, and $w \in V'_{i-1}\}$; $i := i-1$
end
until $i = 1$; **comment** $V_k = V'_k$, $V_{k-1} = V'_{k-1}, \ldots, V_1 = V'_1$ and $V_1, V_2, \ldots, V_k$ are the $s(n)$-partition's sets of T
end;

It remains to describe the procedure GTTRANSFORMATION which appears in the above algorithm.

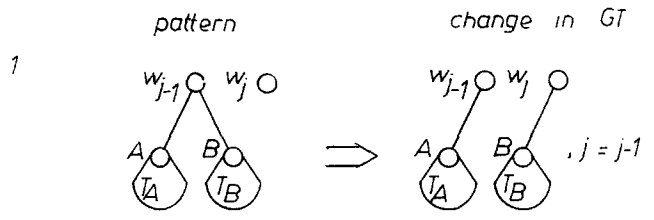
The procedure GTTRANSFORMATION transforms GT (simultaneously ST) so that a virtual son of w_j (the variable j is global for GTTRANSFORMATION) is replaced by a vertex from T.

This process will proceed by matching to w_j one from given patterns and execute a change in GT connected with this pattern. According to the need, the variable j will be updated.

The patterns and changes are presented in Fig. 4. The patterns divide into:
- those in which both sons of w_j are virtual,
- those in which only the right son of w_j is virtual.

A. Both sons of w_j are virtual. Patterns in this class divide into patterns in which
- w_{j-1} and w_j occur on the same level of ST (Fig. 4(1)).
- w_j occurs one level lower than w_{j-1} and it is not a son of w_{j-1} (Fig. 4(2), (3), (4)).
- w_j is the right son of w_{j-1} in ST (Fig. 4(5), (6), (7)).

B. The right son of w_j is virtual (Fig. 4(8), (9), (10)).



Remarks: (a) Virtual vertices are not drawn.

(b) If X is a vertex in GT, then T_X denotes the subtree of GT rooted at X and of vertices from T (virtual vertices do not belong to T_X).

(c) If after the execution of change in GT the right son of w_{j-1} is virtual, then GTTRANSFORMATION must now be called for w_{j-1} (j is decreased by one).

(d) Remarks (a), (b), (c) refer to remaining, presented patterns.

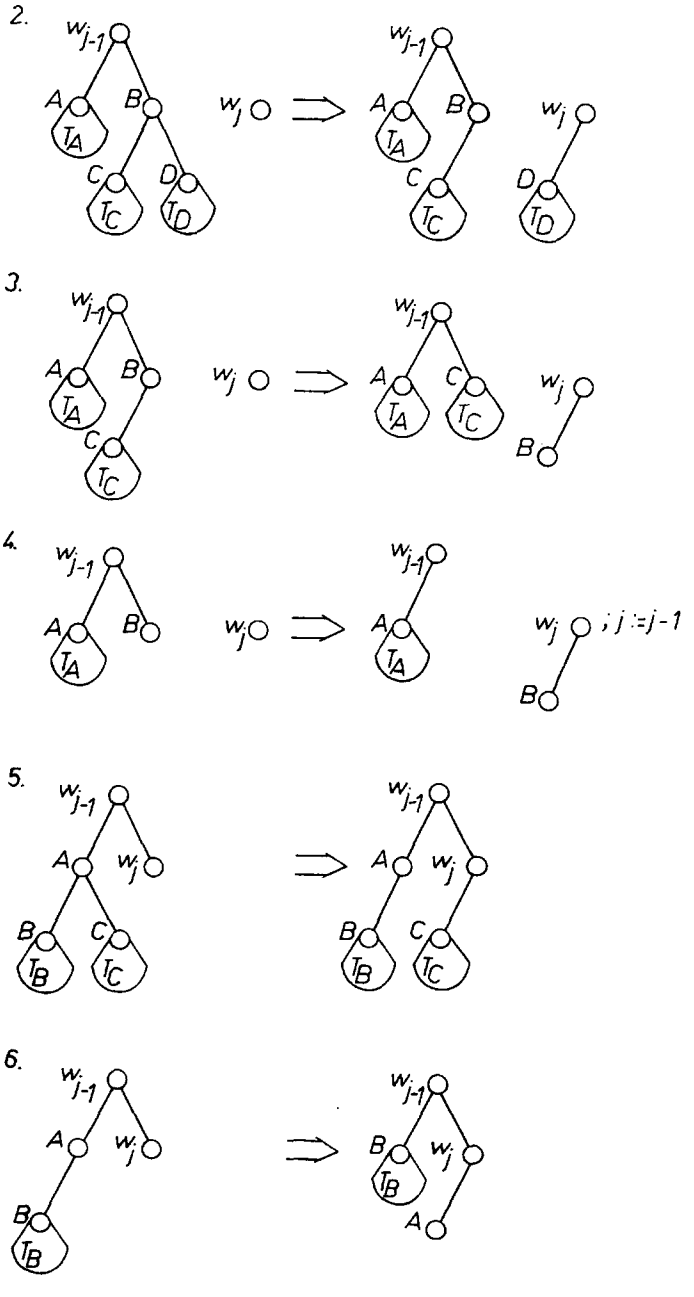


Fig. 4. The patterns and connected with them changes.

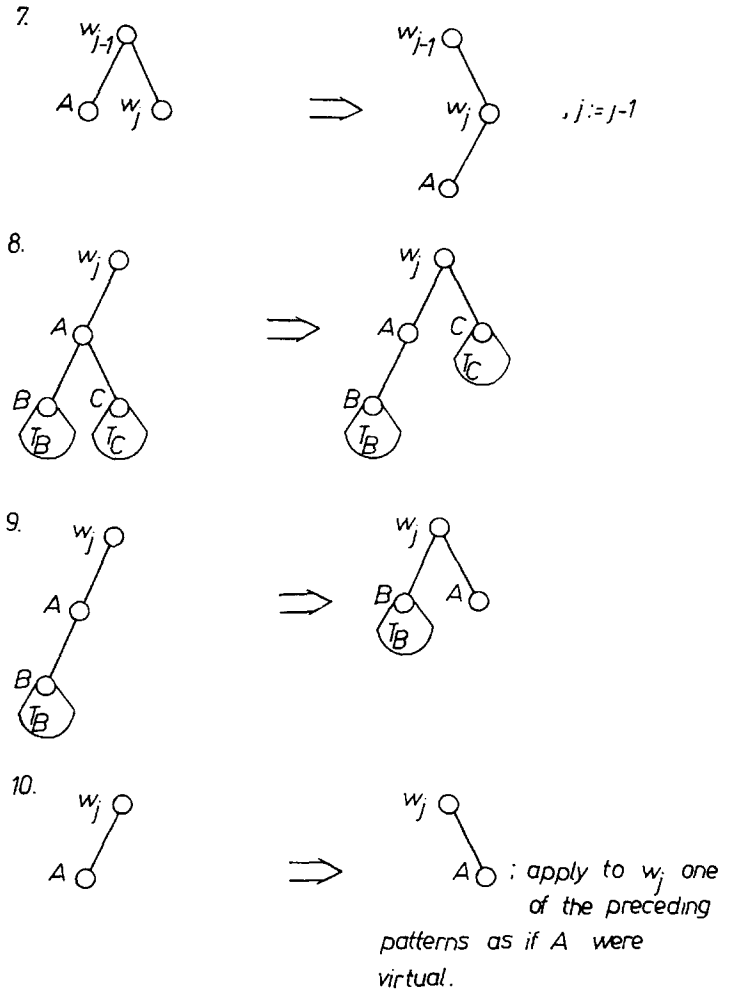


Fig. 4 (*continued*).

Now we may write the procedure GTTRANSFORMATION as follows:

procedure GTTRANSFORMATION;
begin
match to w_j one from the given patterns and execute a change in GT connected with it
end;

We shall analyse the find-$s(n)$-partition algorithm. Therefore, we shall introduce several additional denotations.

(a) For each m, $1 \leq m \leq l$, let T_{w_m} denote a subtree of GT rooted at w_m and only of vertices from T (without virtual vertices) and not containing w_{m+1} (if w_{m+1} is the right son of w_m in GT). We shall denote the number of vertices in T_{w_m} by t_{w_m}.

(b) For each m, $1 \leq m \leq l$, let T'_{w_m} denote a subtree of ST rooted at w_m and not containing w_{m+1} (if w_{m+1} is the right son of w_m).

We shall denote the number of vertices in T'_{w_m} by t'_{w_m}. Let us consider the following conditions.

Condition I. For each m, $1 \leq m \leq l-1$,

$$1 \geq \text{depth}_{ST}(w_{m+1}) - \text{depth}_{ST}(w_m) \geq 0.$$

Condition II. For each m, $2 \leq m \leq l$,

$$\sum_{s=m}^{l} t_{w_s} \leq \sum_{s=m}^{l} t'_{w_s} \quad \text{and} \quad \sum_{s=1}^{l} t_{w_s} = \sum_{s=1}^{l} t'_{w_s}.$$

Condition III. For each m, $1 \leq m \leq l$, for each vertex v in T_{w_m} the number of vertices in the right subtree of v is not greater than the number of vertices in the left subtree of v.

Lemma 4.1. *If Conditions* I, II, III *hold before calling of* GTTRANSFORMATION, *then after its calling: for each pattern matching, after an execution of the change in* GT *connected with it Conditions* I, II, III *hold too.*

Proof. Let us assume that Conditions I, II, III are satisfied before calling GTTRANSFORMATION. The procedure matches a pattern to w_j and executes the change in GT connected with it. We shall prove this lemma only in the case, when a matched pattern is of form 1. The proof is analogous in the remaining cases. It is obvious that, after an execution of a change, Conditions I and III hold. Now we prove that Condition II is true too. Let t_A, t_B denote the numbers of vertices, respectively in T_A, T_B.

Observe that, after the change in GT, $\sum_{s=m}^{l} t_{w_s}$ does not change its value for each m such that $1 \leq m \leq l$ and $m \neq j$:

- for $m > j$ because $t_{w_{j-1}}$ and t_{w_j} do not occur in this sum,
- for $m \leq j-1$ we note that, before the change,

$$\sum_{s=m}^{l} t_{w_s} = \sum_{s=m}^{j-2} t_{w_s} + \underbrace{t_A + t_B + 1}_{\substack{t_{w_{j-1}} \text{ before} \\ \text{the change}}} + \underbrace{1}_{\substack{t_{w_j} \text{ before} \\ \text{the change}}} + \sum_{s=j+1}^{l} t_{w_s}$$

$$= \sum_{s=m}^{j-2} t_{w_s} + \underbrace{t_A + 1}_{\substack{t_{w_{j-1}} \text{ after} \\ \text{the change}}} + \underbrace{1 + t_B}_{\substack{t_{w_j} \text{ after} \\ \text{the change}}} + \sum_{s=j+1}^{l} t_{w_s}$$

$$= \sum_{s=m}^{l} t_{w_s \text{ (after the change)}}$$

To finish the proof it is sufficient to show that, after the change,

$$\sum_{s=j}^{l} t_{w_s} \leq \sum_{s=j}^{l} t'_{w_s},$$

in other words, that

$$\underbrace{t_B+1}_{\substack{t_{w_j} \text{ after} \\ \text{the change}}} + \sum_{s=j+1}^{l} t_{w_s} \leqslant t'_{w_j} + \sum_{s=j+1}^{l} t'_{w_s}.$$

Let us assume that our statement is false. This means that

$$t_B+1+\sum_{s=j+1}^{l} t_{w_s} > t'_{w_j} + \sum_{s=j+1}^{l} t'_{w_s}.$$

But because

$$\sum_{s=j+1}^{l} t_{w_s} \leqslant \sum_{s=j+1}^{l} t'_{w_s},$$

$t_B+1 > t'_{w_j}$.

Hence, and from the facts that $t_A \geqslant t_B$ (Condition III) and $t'_{w_j} = t'_{w_{j-1}}$ we obtain

$$\underbrace{t_A+t_B+1}_{t_{w_{j-1}} \text{ before change}} + \underbrace{1}_{\substack{t_{w_j} \\ \text{before change}}} + \sum_{s=j+1}^{l} t_{w_s} > \sum_{s=j-1}^{l} t'_{w_s}.$$

This contradicts the assumption that Condition II was true before the change. Hence this lemma is true. □

Lemma 4.2. *Let us assume that*

(1) *Conditions* II *and* III *hold,*

(2) w_1 *has a virtual son.*

Then the tree T_{w_1} *has one from the two forms shown in Fig.* 5.

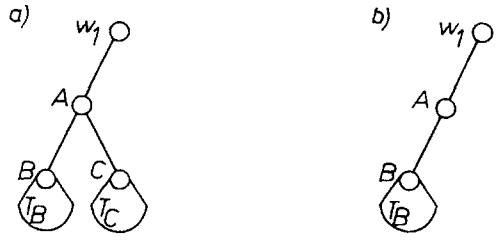


Fig 5. Forms of T_{w_1}.

Proof. It follows from Condition II that $t_{w_1} \geqslant t'_{w_1} \geqslant 3$. From Condition III we obtain that the right son of w_1 must be virtual.

From the above remarks we conclude that the lemma is true. □

Lemma 4.3. *Let us assume that Conditions* I, II, III *are satisfied before an execution of a* **while***-statement (point* B*) of the algorithm. Then, after its execution, sons of* w_1, $w_2, \ldots, w_l$ *are vertices from the tree T.*

Proof. The **while**-statement is controlled by the variable j. First, $j=1$. It follows from Lemma 4.2 that if w_1 has a virtual son, then it is possible to match to w_1 one from the patterns 8, 9 and to execute the change in GT connected with it.

If no son of w_j is virtual, then the value of j is increased by 1. Hence we obtain that if GTTRANSFORMATION is called with $j>1$, then w_{j-1} has not virtual sons. It follows from the above remarks, Lemma 4.1, and Condition I that it is possible to match to w_j one from the patterns $1, 2, 3, \ldots, 10$ and to execute the change in GT. Note that no change in GT executed by GTTRANSFORMATION increases the total number of virtual sons of $w_1, w_2, \ldots, w_l$. The total number of virtual sons of $w_1, w_2, \ldots, w_l$ does not change only in changes connected with the patterns 1, 4, 7. But these changes cause w_{j-1} to obtain a virtual son and the value of j then decreases by 1. The value of j may be decreased only to 1. It follows from Lemmas 4.1 and 4.2 that it will be possible to match to w_1 a pattern and to execute a change in GT that decreases the number of virtual sons of $w_1, w_2, \ldots, w_l$. We conclude from the above considerations that in some moment no son of $w_1, w_2, \ldots, w_l$ will be virtual and the execution of the **while**-statement will stop. □

Lemma 4.4. *Let us assume that*

- *Conditions* I, II, III *are satisfied,*
- *for some* m, $1 \leq m \leq l$, w_m *has no virtual sons.*

Then

(1) *the tree* T_{w_m} *has one of the forms presented in Fig.* 6.

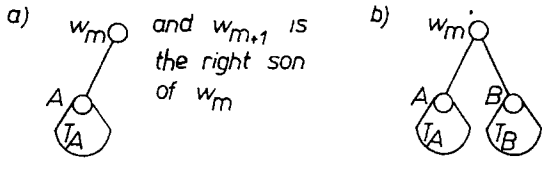


Fig. 6. Forms of T_{w_m}.

(2) *Conditions* II, III *hold for vertices*

- $w_1, w_2, \ldots, w_{m-1}, A, w_{m+1}, \ldots, w_l$ *in case* (a),
- $w_1, w_2, \ldots, w_{m-1}, A, B, w_{m+1}, \ldots, w_l$ *in case* (b).

Proof. (1) is obvious. In order to prove (2) it is sufficient to note that

- $t_{w_m} = t_A + 1$ and $t'_{w_m} = t'_A + 1$ in case (a),
- $t_{w_m} = t_A + t_B + 1$ and $t'_{w_m} = t'_A + t'_B + 1$ and $t_A \geq t_B$ in case (b). □

Lemma 4.5. *After an execution of the find-*$s(n)$*-partition algorithm* $V_1, V_2, \ldots, V_k$ *are the* $s(n)$*-partition's sets of* T.

Proof. Note that in the find-$s(n)$-partition algorithm the changing tree ST is searched with the BFS method, starting at the root R, modified as follows. Successively reached vertices are first, vertices only from V'_k, next from $V'_{k-1}, \ldots, V'_1$. If a vertex

w is reached and if it belongs to V_i', then it will belong to V_i from now on (point C of algorithm). Point B of the algorithm causes that each V_i, $1 \leq i \leq k$, will contain only vertices from T. In order to prove this, it is sufficient to notice that before any execution of B the assumptions of Lemma 4.3 are satisfied. But observe that after the first entry to a **repeat**-statement of algorithm $l = 1$, $w_1 = R$ and the assumptions of Lemma 4.3 hold. It follows from Lemmas 4.1 and 4.3 that B keeps Conditions I, II, III and after its termination no son $w_1, w_2, \ldots, w_l$ is virtual. The vertices $w_1, w_2, \ldots, w_l$ change in consequence of C.

We conclude from Lemma 4.4 and Corollaries 3.3–3.5 that new $w_1, w_2, \ldots, w_l$ will again satisfy the assumptions of Lemma 4.3. Next we notice that if a vertex of T belongs to V_i, then it does not change a position in ST.

From the above considerations we obtain that after an execution of the algorithm the sets $V_1, V_2, \ldots, V_k$ are the partition's sets of vertices of T.

To finish the proof, let us observe that if v is the father of w in the tree T and $v \in V_i$, then w can only belong to $V_{i-1} \cup V_i \cup V_{i+1}$. Let us consider two cases to prove the above fact.

Case (a) After an execution of the find-$s(n)$-partition algorithm, v is the father of w in GT. It follows from the construction of the $s(n)$-partition of GT that w can belong only to V_i or V_{i+1}.

Case (b) After an execution of the find-$s(n)$-partition algorithm, v is not the father of w in GT. Let us consider the moment when v ceases to be the father of w in GT. There are again two cases:

Case (b1) v is equal to some w_m from V_i. To T_{w_m} is applied one from the changes 1, 3, 4, 6, 7 after which w ceases to be the son of v. But after this change w is the son of some other vertex from V_i. Sons of the vertices from V_i belong either to V_i or to V_{i+1}.

Case (b2) v is the son of some w_m from V_i or V_{i-1}. One of the changes 2, 3, 5, 6, 8, 9, after which w ceases to be the son of v, is applied to T_{w_m}. But, after this change, w is the son of some other vertex from V_i or V_{i-1} according to whether $w_m \in V_i$ or $w_m \in V_{i-1}$.

Hence w can belong only to $V_{i-1} \cup V_i \cup V_{i+1}$. □

We now finish this paper with the following theorem.

Theorem 4.6. *Any n-vertex binary tree can be embedded in L_n with dilation cost* $O(n/\log_2 n)$.

Proof. Let T be an arbitrary tree from $\mathcal{T}_n$ and let m be the natural number defined as follows:

$$m = \min_{h \geq 0} \{2^{h+1} - 1 : (2^{h+1} - 1 - n) \geq 0\}.$$

We add to T $m - n$ vertices to obtain m-vertex binary tree. (T must be a subtree of this augmented tree.) Let us denote this tree by T'. Next we transform T' in such

way that for each vertex v of T' the number of vertices in its right subtree will ıe at most equal to the number of vertices in its left subtree. To transformed T' we ıpply the find-$s(n)$-partition algorithm. It follows from Lemma 4.5 that there is an (m)-partition of T'. If from sets of this $s(m)$-partition we remove added vertices, hen we obtain an $s(m)$-partition of T. Because $s(m) \leqslant 2s(n)$, from Lemmas 2.1 nd 3.1 we conclude that the claim of the theorem is true. □

;. Conclusions

In this paper we have shown that any n-vertex binary tree can be embedded in L_n with dilation cost $O(n/\log_2 n)$. It is easy to implement the find-$s(n)$-partition ılgorithm in $O(n^2/\log_2 n)$ time. We do not do this here, because what is really nteresting is to find a fast algorithm that embeds any binary tree $T \in \mathcal{T}_n$ in L_n with lilation cost equal to $c*\text{DIL}(T, L_n)$ ($\text{DIL}(T, L_n)$ denotes $\min_F \text{DIL}_F(T, L_n)$), where : is the constant independent of T and n.[2] The author hopes that the techniques ısed in this paper may be helpful for that purpose.

\cknowledgment

I would like to thank L. Banachowski for his helpful remarks and suggestions.

?eferences

1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).
2] M.K. Garey, R.L. Graham, D.S. Johnson and D.E. Knuth, Complexity results for bandwidth minimization, *SIAM J. Appl. Math.* **43** (3) (1978) 477-495.
3] A.L. Rosenberg, Issues in the study of graph embeddings, in: *Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science **100** (Springer, Berlin, 1980).
4] A.L. Rosenberg and L. Snyder, Bounds on the costs of data encodings, *Math. Systems Theory* **12** (1978) 9-39.

[2] The problem to determine for an n-vertex binary tree T and an integer k if $\text{DIL}(T, L_n) \leqslant k$ is NP-complete [2].